

Práctica 2. Sistemas basados en reglas en Robocode

Inteligencia Artificial 2012/13

FJRP, VMDB, SFL – CCIA-2012

Octubre-2012

Índice

1. Descripción de la práctica	1
1.1. Objetivos	1
1.2. Tareas a realizar	1
1.3. Normas de entrega	2
2. Descripción del problema	2
3. Herramientas a utilizar	2
4. Código de partida	3
4.1. Método run() de RobotDrools	4
4.2. Uso del código aportado e integración en Robocode	5
4.2.1. Ficheros incluidos	5
4.2.2. Compilación y uso	6
4.2.3. Verificación del fichero de reglas	7

1. Descripción de la práctica

1.1. Objetivos

- Conocer y utilizar el motor de reglas JBoss Drools Expert
- Experimentar con reglas de control para un robot "de juguete"

1.2. Tareas a realizar

Se deberán implementar dos juegos de reglas DROOLS que definan el comportamiento de dos robots diferentes.

Opcionalmente se puede plantear una pequeña comparativa informal del comportamiento de los robots implementados ante batallas y enemigos equivalentes.

1.3. Normas de entrega

- La práctica se realizará de forma individual o en parejas.
- La **fecha límite** de entrega para la convocatoria de febrero será el **xx/12/2012** en el despacho 303.
- Antes de esa fecha es posible entregar la práctica en horario de prácticas o de tutorías.
- Se deberá entregar:
 - Código fuente: reglas creadas + clases Java adicionales (preferiblemente en CD o en USB para copiarlo)
 - Memoria en papel con la estructura que se indica
- En el momento de la entrega se comprobará el funcionamiento de la práctica y el modo en que fue desarrollada, por lo que, salvo causas de fuerza mayor, será necesaria la presencia de todos los miembros del grupo.
- Estructura de la memoria (aproximadamente 5-8 páginas):
 1. Descripción breve del problema y de las herramientas utilizadas
 2. Descripción de los dos robots implementados. Para cada uno:
 - Descripción breve de la estrategia que se pretende que guíe al robot
 - Descripción textual de cada una de las reglas implementadas junto con su código.
 - Descripción de los elementos adicionales (definición de nuevos tipos de hechos, reglas de apoyo, etc)
 3. Descripción del código adicional aportado (en caso de que exista)
 - Describir las clases Java y los métodos añadidos sobre el código de partida
 4. Experimentos realizados (en caso de llevarlos a cabo) y resultados obtenidos
 5. Conclusiones y problemas encontrados
 6. Bibliografía consultada (libros, enlaces, código, ...)

Nota: Incluir Nombre, DNI y e-mail en la portada. No es necesario ningún tipo especial de encuadernación.

2. Descripción del problema

Se trata de construir un robot para el entorno de simulación de batallas Robocode cuyo control se realice mediante un conjunto de reglas de producción ejecutadas sobre el motor de reglas JBoss Drools Expert.

Se deberán implementar y documentar dos juegos de reglas para dos tipos de robots

3. Herramientas a utilizar

Información sobre Robocode:

- Página principal de Robocode (descargas, documentación y ejemplos)
 - Javadoc

- Robowiki
- Robocode Argentina (tutoriales muy completos)
 - **Guia de programación:** Tutorial de programación
Incluye un resumen bastante completo sobre los eventos manejados por los robots de Robocode muy útil para la realización de la práctica.
 - **Manejo de ángulos y posiciones:** Tutorial de programación
- Página de Manuel Lucena (U. Jaen) (réplica de los tutorial de Robocode Argentina)
 - **Guia de programación:** Tutorial de programación
 - **Manejo de ángulos y posiciones:** Tutorial de programación

Información sobre JBoss Drools Expert

- Página principal de JBoss Drools Expert
 - Descarga (Drools 5.4.0 Final)
Copia local: drools-distribution-5.4.0.Final.zip
 - Manual de usuario Drools Expert
Prestad especial atención al capítulo 5, "The Rule Language", (en concreto a la sección 5.8), donde se explica el lenguaje de definición de reglas DRL

4. Código de partida

Se partirá de un esqueleto de Robot (archivo `RobotDrools.java`) que delega la gestión de sus eventos y la decisión de acciones al motor de reglas Drools.

- El *RobotDrools* es de tipo *AdvancedRobot*, lo que permite ejecutar distintas acciones a la vez.
- El método *run()* es llamado por el framework Robocode y, junto con los manejadores de eventos, contiene el código Java que controla el comportamiento del robot
- En este caso en método *run()* delega las decisiones al motor de reglas, realizando el siguiente ciclo en cada turno
 1. Todos los eventos recibidos desde el turno anterior ya habrán sido capturados por sus respectivos métodos gestores (*onScannedRobot()*, *onHitByBullet()*, ...) e incorporados a la *Memoria Activa* de DROOLS (objeto *ksession* de tipo *StatefulKnowledgeSession*) del motor de reglas
 - Los tipos de evento a disposición de DROOLS y susceptibles de ser utilizados en las reglas son:
 - `ScannedRobotEvent`
 - `HitByBulletEvent`
 - `HitRobotEvent`
 - `HitWallEvent`
 - `BulletHitBulletEvent`
 - `BulletHitEvent`
 - `BulletMissedEvent`
 - `RobotDeathEvent`
 - Los eventos producidos en cada turno estarán disponibles en la memoria activa y podrán incluirse en el lado izquierdo de las reglas usando la sintaxis de DROOLS y accediendo a las propiedades disponibles (ver en los respectivos JavaDoc los métodos *getPROPIEDAD()* para ver con que propiedades cuenta cada evento).

2. Incorpora a la memoria activa un objeto *EstadoRobot* con el estado actual del robot (posición, energía, etc)
 - El objeto *EstadoRobot* ofrece las mismas propiedades que el objeto de Robocode *RobotStatus*
 - Por comodidad se incluyen tres métodos adicionales calculados a partir de los anteriores
 - *getPosicion()*: encapsula en un objeto *Posicion* las coordenadas actuales del robot
 - *getAnguloRelativoCanon()*: proporciona el ángulo relativo del cañón respecto al ángulo absoluto del tanque (desplazamiento, entre -180 y 180 grados, respecto a la dirección de avance)
 - *getAnguloRelativoRadar()*: proporciona el ángulo relativo del radar respecto al ángulo absoluto del tanque (desplazamiento, entre -180 y 180 grados, respecto a la dirección de avance)
 3. Incorpora a la memoria activa un objeto *EstadoBatalla* con el estado actual del entorno de Robocode (núm. de oponentes, tamaño del campo de batalla, etc)
 4. Lanza el proceso de inferencia de DROOLS pidiendo la ejecución de todas las reglas activadas por los hechos actuales
 5. Elimina de la memoria activa los hechos actuales (eventos, estado robot, etc)
 6. Recupera los hechos de tipo *Accion* insertados por las reglas ejecutadas e inicia su ejecución
 7. Pasa el control a Robocode con el método *execute()*, indicándole que el Robot ha terminado su turno
- El comportamiento de este robot está determinado por las reglas presentes en el fichero de reglas '**reglas_robot.drl**' que se carga en el motor de inferencias nada más iniciar el robot.
- Dichas reglas serán activadas por los eventos introducidos en la memoria activa por los gestores de eventos y por los objetos *EstadoRobot* y *EstadoBatalla* introducidos por el método *run()*
 - Las posibles acciones a realizar se encapsulan en hechos/objetos de tipo *Accion*, donde se indica el tipo de acción junto con un parámetro de tipo *double* para cada acción y su respectiva prioridad
- Tipos de acciones estándar considerados:
- *Accion.AVANZAR*: avanza el núm. de pixels indicados por *parámetro* (invocará el método *setAhead(parámetro)*)
 - *Accion.RETOCEDER*: retrocede el núm. de pixels indicados por *parametro* (invocará el método *setAhead(parámetro)*)
 - *Accion.STOP*: detiene el movimiento del robot, no tiene en cuenta el valor de *parámetro* (invocará el método *setStop()*)
 - *Accion.DISPARAR*: dispara con la intensidad indicada por *parametro* (invocará el método *setFire(parámetro)*)
 - *Accion.GIRAR_TANQUE_DER*: gira el tanque hacia la derecha el núm. de grados indicados por *parametro* (invocará el método *setTurnRight(parámetro)*)
 - *Accion.GIRAR_TANQUE_IZQ*: gira el tanque hacia la izquierda el núm. de grados indicados por *parametro* (invocará el método *setTurnLeft(parámetro)*)
 - *Accion.GIRAR_RADAR_DER*: gira el radar hacia la derecha el núm. de grados indicados por *parametro* (invocará el método *setTurnRadarRight(parámetro)*)
 - *Accion.GIRAR_RADAR_IZQ*: gira el radar hacia la izquierda el núm. de grados indicados por *parametro* (invocará el método *setTurnRadarLeft(parámetro)*)
 - *Accion.GIRAR_CANON_DER*: gira el cañón hacia la derecha el núm. de grados indicados por *parametro* (invocará el método *setTurnGunRight(parámetro)*)
 - *Accion.GIRAR_CANON_IZQ*: gira el cañón hacia la izquierda el núm. de grados indicados por *parametro* (invocará el método *setTurnGunLeft(parámetro)*)

4.1. Método *run()* de *RobotDrools*

```
package drools_robocode;
...
public class RobotDrools extends AdvancedRobot {
    ...
```

```

public void run() {
    DEBUG.habilitarModoDebug(System.getProperty("robot.debug", "true").equals("true"));

    // Crear Base de Conocimiento y cargar reglas
    crearBaseConocimiento();

    // Hacer que movimiento de tanque, radar y cañon sean independientes
    setAdjustGunForRobotTurn(true);
    setAdjustRadarForGunTurn(true);
    setAdjustRadarForRobotTurn(true);

    while (true) {
        DEBUG.mensaje("inicio turno");
        //cargarEventos(); // se hace en los métodos onXXXXXEvent()
        cargarEstadoRobot();
        cargarEstadoBatalla();

        // Lanzar reglas
        DEBUG.mensaje("hechos en memoria activa");
        DEBUG.volcarHechos(ksession);
        ksession.fireAllRules();
        limpiarHechosIteracionAnterior();

        // Recuperar acciones
        Vector<Accion> acciones = recuperarAcciones();
        DEBUG.mensaje("acciones resultantes");
        DEBUG.volcarAcciones(acciones);

        // Ejecutar Acciones
        ejecutarAcciones(acciones);
        DEBUG.mensaje("fin turno\n");
        execute(); // Informa a robocode del fin del turno (llamada bloqueante)
    }

    }
    ...
} // Fin RobotDrools

```

4.2. Uso del código aportado e integración en Robocode

Descarga: Código de partida drools_robocode.tar.gz (incluye Javadoc)

4.2.1. Ficheros incluidos

drools_robocode	
RobotDrools.java:	implementación del robot guiado por DROOLS
EstadoBatalla.java:	encapsula la información sobre el estado actual del campo de batalla
EstadoRobot.java:	encapsula la información sobre el estado actual del robot
Accion.java:	encapsula las posibles acciones a realizar por el robot
Posicion.java:	encapsula las coordenadas X e Y de una posición del campo de batalla
Auxiliar.java:	funciones auxiliares para calulo de ángulos, posiciones, distancias basadas en http://www.robocode-argentina.com.ar/descargas/trig.doc (en caso de necesitar implementar funciones adicionales este es el lugar más cómodo para hacerlo)

ComprobarReglas.java: clase cargadora de ficheros de reglas, permite comprobar que los ficheros de reglas se compilan sin errores sin tener que lanzar el robot y realizar pequeñas pruebas insertando hechos
''a mano''

DEBUG.java: clase con métodos para mostrar mensajes de depuración

drools_robotcode/reglas (directorio de reglas)

reglas_robot.drl: fichero de reglas de ejemplo, implementa un robot trazador simple

4.2.2. Compilación y uso

1. Pasos previos

- Descargar e instalar Robocode (Jar autoinstalable: `java -jar robocode-1.7.4.2-setup.jar`)
- Descargar y descomprimir DROOLS

2. Establecer los directorio de instalación en un par de variables de entorno

```
export ROBOCODE_PATH=/home/[...]/robocode/  
export DROOLS_PATH=/home/[...]/drools-distribution-5.4.0.Final/binaries/
```

3. Copiar el directorio del robot (drools_robotcode) en directorio robots de la instalación de Robocode

```
tar xzvf drools_robotcode.tar.gz  
cp -r drools_robotcode $ROBOCODE_PATH/robots/
```

4. Establecer los ficheros .jar necesarios en una variable de entorno

```
export JARS=$ROBOCODE_PATH/libs/robocode.jar  
export JARS=$JARS:$DROOLS_PATH/drools-core-5.4.0.Final.jar:$DROOLS_PATH/drools-compiler-5.4.0.Final.jar  
export JARS=$JARS:$DROOLS_PATH/knowledge-api-5.4.0.Final.jar;$DROOLS_PATH/binaries/knowledge-internal-api-5.4.0.Final.jar  
export JARS=$JARS:$DROOLS_PATH/ecj-3.5.1.jar:$DROOLS_PATH/antlr-runtime-3.3.jar  
export JARS=$JARS:$DROOLS_PATH/mvel2-2.1.0.drools16.jar
```

Nota: indicad las versiones concretas de vuestra instalación

5. Compilar el robot (necesario la primera vez y siempre que se cambie o añada código java al paquete drools_robotcode)

```
javac -cp $JARS:. $ROBOCODE_PATH/robots/drools_robotcode/*.java
```

6. Arrancar Robocode con las opciones necesarias para integrar el robot DROOLS

```
java -Xmx512M -Dsun.io.useCanonCaches=false -DNOSECURITY=true -DWORKINGDIRECTORY=$ROBOCODE_PATH \  
-Drobot.reglas=$ROBOCODE_PATH/robots/drools_robotcode/reglas/reglas_robot.drl -Drobot.debug=true \  
-cp $JARS:. robocode.Robocode
```

Mediante la propiedad `robot.reglas` se indica el path absoluto del fichero de reglas (.drl) a utilizar

- si no se indica esta propiedad se cargarán las reglas del fichero indicado en la constante `FICHERO_REGLAS` de la clase `RobotDrools.java` que por defecto utiliza el fichero `$ROBOCODE_PATH/robots/drools_robotcode/reglas_robot.drl`

• Importante:

- En el caso de enfrentar a dos o más robots basados en DROOLS siguiendo el esquema utilizado ambos robots usarían el mismo fichero de reglas.
- Se deberán crear dos paquetes distintos y establecer el fichero de reglas de cada robot en la constante `FICHERO_REGLAS` de las clases `RobotDrools.java` respectivas o definir un nombre de *property* específico para cada robot.

Mediante la propiedad `robot.debug` se indica si se mostrarán o no los mensaje de DEBUG en la consola del robot

El robot *RobotDrools* deberá aparecer en el grupo *drools_robotcode*

7. Compilación y uso en MS Windows XP: script-windows.bat

Nota: Antes de ejecutar el fichero `.bat` ajustad las variables `ROBOCODE_PATH` y `DROOLS_PATH` a los valores que correspondan.

4.2.3. Verificación del fichero de reglas

Se incluye una clase `ComprobarReglas.java` que permite ejecutar el compilador de reglas de DROOLS sin tener que cargar el robot en el entorno Robocode. También permite insertar hechos en la memoria activa "a mano" desde el código de `ComprobarReglas.java` (requiere recompilar, ver paso 5)

Desde el directorio `$ROBOCODE_PATH/robots/`

```
java -Drobot.reglas=[PATH ABSOLUTO A FICHERO_REGLAS.drl] -Drobot.debug=true \  
-cp $JARS:. drools_robotcode.ComprobarReglas
```